

Лабораторная работа 2

Сортировки

Финоченко Александр Викторович Б02-201

Цель работы: написать улучшения к сортировкам с квадратичной асимптотической сложностью, провести декомпозицию на элементарные шаги и протестировать отдельные функции.

Шейкерская сортировка

Ниже приведены функции прямого прохода, обратного прохода и шейкерской сортировки.

```
1 bool forward_step(U arr[], U const begin_idx, U const end_idx) {
2     bool sorted = true;
3     for (U idx = begin_idx; idx != end_idx; ++idx) {
4         if (arr[idx] > arr[idx + 1]) {
5             auto tmp = arr[idx];
6             arr[idx] = arr[idx + 1];
7             arr[idx + 1] = tmp;
8             sorted = false;
9         }
10    }
11    return sorted;
12 }
13
14 bool backward_step(U arr[], U const begin_idx, U const end_idx) {
15     bool sorted = true;
16     for (U idx = end_idx - 1; idx != begin_idx - 1; --idx) {
17         if (arr[idx] > arr[idx + 1]) {
18             auto tmp = arr[idx];
19             arr[idx] = arr[idx + 1];
20             arr[idx + 1] = tmp;
21             sorted = false;
22         }
23    }
24    return sorted;
25 }
26
27 void shaker_sort(U arr[], U const begin_idx, U const end_idx) {
28     bool sorted = false;
29     while (!sorted) {
30         sorted = forward_step(arr, begin_idx, end_idx);
31         sorted = backward_step(arr, begin_idx, end_idx);
32         assert(is_sorted(arr, begin_idx, end_idx));
33     }
34 }
```

Проверка того, что массив отсортирован, производилась с помощью функции `is_sorted`:

```
1 template <typename T>
2 bool is_sorted(T arr[], U const begin_idx, U const end_idx) {
3     bool sorted = true;
4     for (U idx = begin_idx; idx != end_idx; idx++) {
5         if (arr[idx] > arr[idx + 1])
6             sorted = false;
7     }
8     return sorted;
9 }
```

Программа успешно отработала, что означает, что функция сортировки работает нормально.

Сортировка расчёской

Сортировка проводится сначала по $N/4$ и $N/2$, далее проводится обычная сортировка пузырьком. Полный код функции сортировки расчёской представлен ниже

```
1  template <typename T>
2  bool is_N4_sorted(T arr[], U const begin_idx, U const end_idx) {
3      U step = (end_idx - begin_idx + 1) / 4;
4      bool sorted = true;
5      for (U idx = begin_idx; idx + step <= end_idx; idx += step) {
6          if (arr[idx] > arr[idx + step])
7              sorted = false;
8      }
9      return sorted;
10 }
11
12 template <typename T>
13 bool is_N2_sorted(T arr[], U const begin_idx, U const end_idx) {
14     U step = (end_idx - begin_idx + 1) / 2;
15     bool sorted = true;
16     for (U idx = begin_idx; idx + step <= end_idx; idx += step) {
17         if (arr[idx] > arr[idx + step])
18             sorted = false;
19     }
20     return sorted;
21 }
22
23 ULL N4_sort(U arr[], U const begin_idx, U const end_idx) {
24     U step = (end_idx - begin_idx + 1) / 4;
25     ULL count = 0;
26     bool sorted = false;
27     while (!sorted) {
28         sorted = true;
29         for (U idx = begin_idx; idx + step <= end_idx; idx += step) {
30             if (arr[idx] > arr[idx + step]) {
31                 auto tmp = arr[idx];
32                 arr[idx] = arr[idx + step];
33                 arr[idx + step] = tmp;
34                 sorted = false;
35                 count++;
36             }
37         }
38     }
39     assert(is_N4_sorted(arr, begin_idx, end_idx));
40     return count;
41 }
42
43 ULL N2_sort(U arr[], U const begin_idx, U const end_idx) {
44     U step = (end_idx - begin_idx + 1) / 2;
45     ULL count = 0;
46     bool sorted = false;
47     while (!sorted) {
48         sorted = true;
49         for (U idx = begin_idx; idx + step <= end_idx; idx += step) {
50             if (arr[idx] > arr[idx + step]) {
51                 auto tmp = arr[idx];
52                 arr[idx] = arr[idx + step];
```

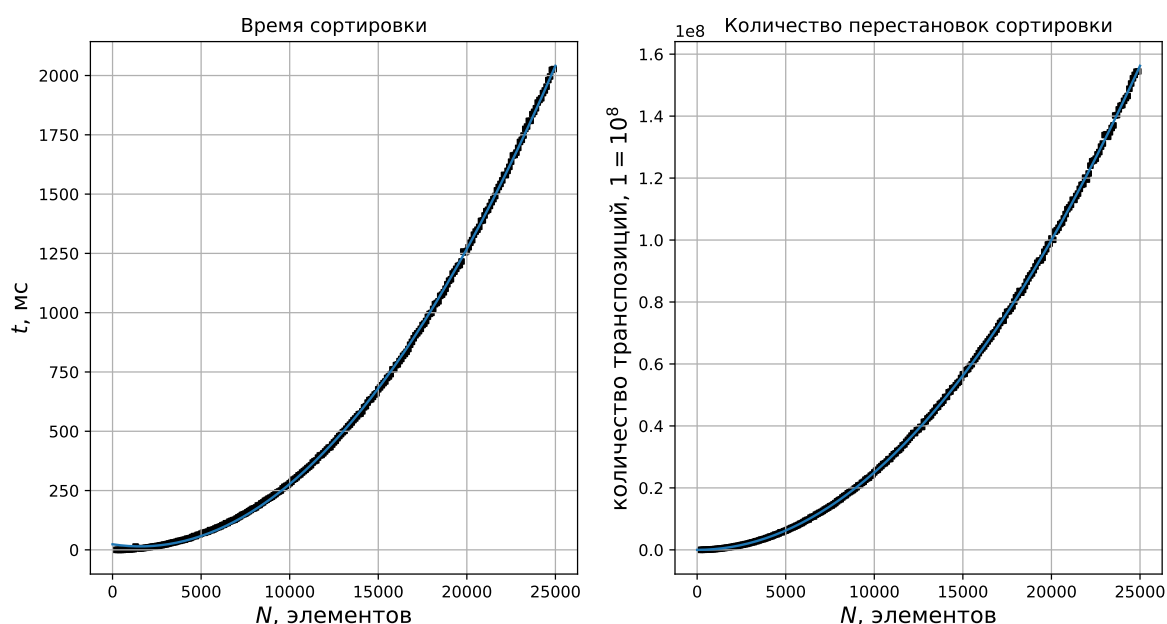
```

53         arr[idx + step] = tmp;
54         sorted = false;
55         count++;
56     }
57 }
58 }
59 assert(is_N2_sorted(arr, begin_idx, end_idx));
60 return count;
61 }
62
63 ULL comb_sort(U arr[], U const begin_idx, U const end_idx) {
64     ULL count1 = N4_sort(arr, begin_idx, end_idx);
65     ULL count2 = N2_sort(arr, begin_idx, end_idx);
66     ULL count = 0;
67
68     bool sorted = false;
69     while (!sorted) {
70         sorted = true;
71         for (U idx = begin_idx; idx != end_idx; ++idx) {
72             if (arr[idx] > arr[idx + 1]) {
73                 auto tmp = arr[idx];
74                 arr[idx] = arr[idx + 1];
75                 arr[idx + 1] = tmp;
76                 sorted = false;
77                 count++;
78             }
79         }
80     }
81     return count1 + count2 + count;
82 }

```

Программа была проверена на случайно сгенерированных массивах и ошибок не выдавала. Сама функция одновременно сортирует массив и подсчитывает количество перестановок.

Далее представлен график зависимости времени и числа перестановок в зависимости от количества элементов N



Видно, что зависимость квадратичная.

Сортировка Шелла

Общая функция для сортировки Шелла с некоторым шагом

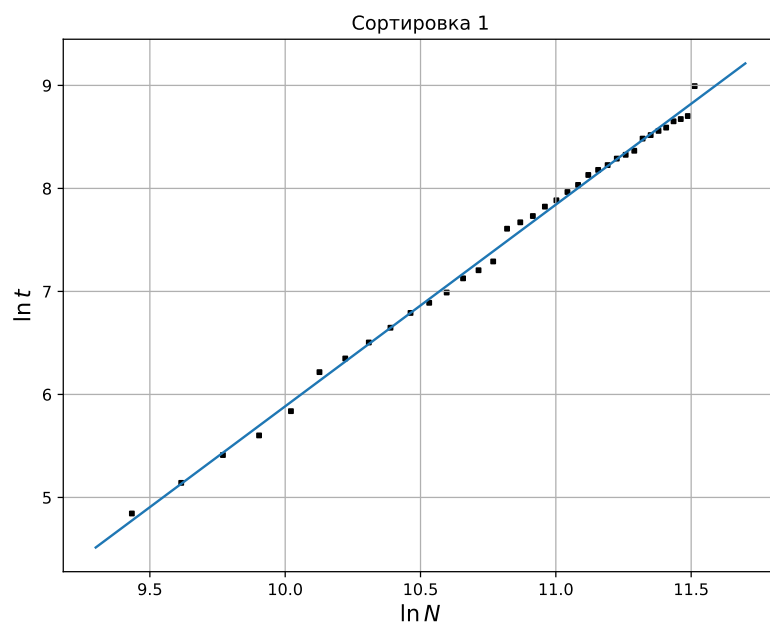
```
1 ULL shell_sort_i(U arr[], U const begin_idx, U const end_idx, U step) {
2     ULL count = 0;
3     if (begin_idx + step > end_idx)
4         return 0;
5     for (int last = end_idx - step; last >= 0; last -= step) {
6         auto tmp_idx = last;
7         while (tmp_idx + step <= end_idx && arr[tmp_idx] > arr[tmp_idx + step]) {
8             auto tmp = arr[tmp_idx];
9             arr[tmp_idx] = arr[tmp_idx + step];
10            arr[tmp_idx + step] = tmp;
11            tmp_idx += step;
12            count++;
13        }
14    }
15    return count;
16 }
```

Последовательность $d_{i+1} = d_i/2, d_1 = N$

Для шага $d_{i+1} = d_i/2, d_1 = N$ функция сортировки Шелла выглядит так

```
1 ULL shell_sort(U arr[], U const begin_idx, U const end_idx) {
2     ULL count = 0;
3     U step = (end_idx - begin_idx + 1);
4
5     while (step != 0) {
6         count += shell_sort_i(arr, begin_idx, end_idx, step);
7         step /= 2;
8     }
9     return count;
10 }
```

Предполагая, что при больших N время работы программы приближается к N^α , определим это α . Для этого перейдём к логарифмам, тогда $\ln t = \alpha \ln N$. Получаем, что $\alpha = 1.96$



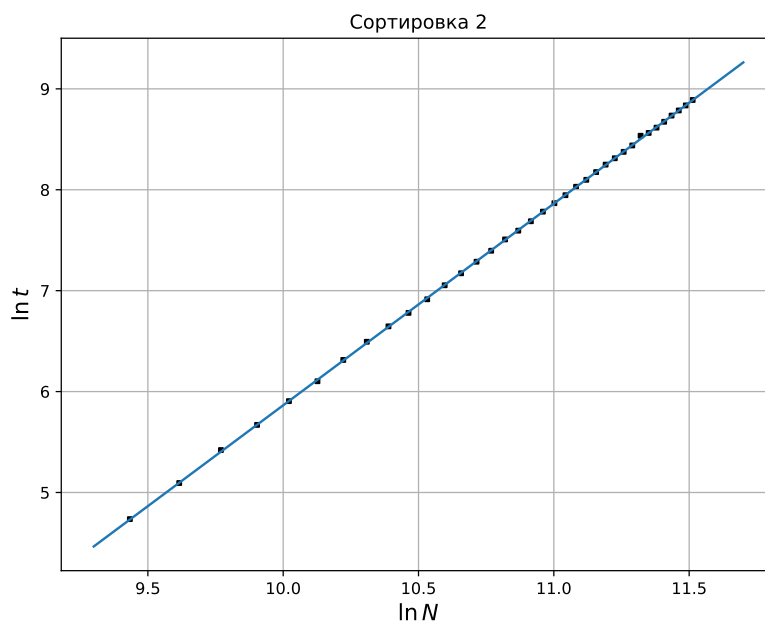
Среднее количество перестановок равно $6.88 \cdot 10^8$.

Последовательности Хиббарда

Прделаем тоже самое. Функция функция сортировки Шелла выглядит так

```
1 ULL shell_sort(U arr[], U const begin_idx, U const end_idx) {
2     ULL count = 0;
3     U i = 0;
4     while ((U)pow(2, i) - 1 <= (end_idx - begin_idx + 1)) {
5         ++i;
6     }
7     --i;
8     U step = (U) pow(2, i) - 1;
9     while (step != 0) {
10        count += shell_sort_i(arr, begin_idx, end_idx, step);
11        i--;
12        step = pow(2, i) - 1;
13    }
14    return count;
15 }
```

Если $\ln t = \alpha \ln N$, то $\alpha = 1.99$



Среднее количество перестановок равно $7.06 \cdot 10^8$.

Обратная последовательность Фибоначчи

Для обратной последовательности Фибоначчи функция сортировки Шелла выглядит так

```
1 ULL shell_sort(U arr[], U const begin_idx, U const end_idx) {
2     ULL count = 0;
3     vector<U> f_arr = fib_arr(end_idx - begin_idx + 1);
4     U step;
5     for (U i = size(f_arr) - 1; i != 1; i--) {
6         step = f_arr[i];
```

```

7     count += shell_sort_i(arr, begin_idx, end_idx, step);
8 }
9 return count;
10 }

```

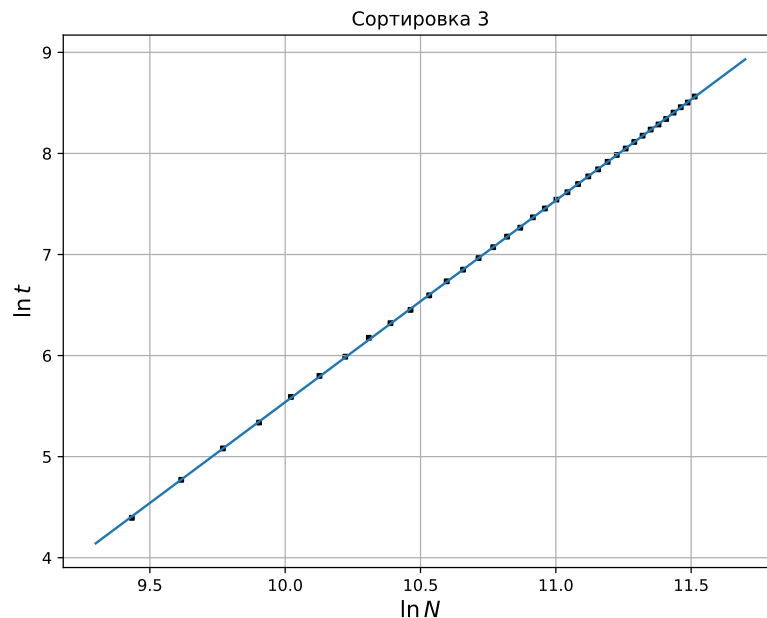
Здесь проще использовать векторы. Функция `fib_arr` нужна, чтобы быстро находить числа Фибоначчи

```

1 vector<U> fib_arr(U N) {
2     U F_0 = 0; U F_1 = 1;
3     U n = 0;
4     while (F_1 <= N) {
5         U temp = F_1;
6         F_1 += F_0;
7         F_0 = temp;
8         n++;
9     }
10    F_0 = 0; F_1 = 1;
11    vector<U> fib_arr(n + 1);
12    fib_arr[0] = 0;
13    for (U i = 1; i <= n; i++) {
14        fib_arr[i] = F_1;
15        U temp = F_1;
16        F_1 += F_0;
17        F_0 = temp;
18    }
19    return fib_arr;
20 }

```

Если $\ln t = \alpha \ln N$, то $\alpha = 1.99$



Среднее количество перестановок равно $5.07 \cdot 10^8$.

Вывод

Получаем, что наиболее эффективная по времени является 1 последовательность, наиболее эффективная по числу перестановок - 2 последовательность.